

Mini-Curso Frontend: CRUD Simples

Autor: [Leonardo Silva](#)

Revisão: Guilherme Marques

Data: abril de 2025

Neste mini-curso Frontend iremos implementar um CRUD Simples com uso do LocalStorage com objetivo de fazermos as instalações e configurações básicas de um projeto frontend que usamos em nosso laboratório.

Este projeto você encontra no link:

<https://github.com/EstudosCpid/mini-curso-frontend-2025>

Vamos lá?

Pré-requisitos de instalação do Quasar Framework

Instalação do Visual Studio Code (VSCode)

- Acesse o site oficial do VSCode (<https://code.visualstudio.com/>)
- Faça o download para o seu sistema operacional.
- Após o download, execute o instalador e siga as instruções para completar a instalação.

Instalação do Node.js e NPM.

- Acesse o site do Node.js (<https://nodejs.org/en>) e baixe a versão recomendada (LTS)
- Execute o instalador e siga as instruções para instalar corretamente o Node e o NPM

Verificação da instalação.

Para verificar se a instalação foi bem-sucedida, abra o terminal do vscode e rode os seguintes comandos:

```
node -v
```

Para verificarmos a versão do Node.js, a versão que estou utilizando é a v22.9.0).

```
leonardosc@CIDIG040:~/Quasar-Collection/crud-simples$ node -v  
v22.9.0
```

E verificamos o gerenciador de pacotes npm com o comando abaixo:

```
npm -v
```

A versão que estou utilizando é a 10.9.2).

```
● leonardosc@CIDIG040:~/Quasar-Collection/crud-simples$ npm -v  
10.9.2
```

Isso irá mostrar a versão do Node e do NPM caso esteja instalado corretamente.

Instalação e configuração do Quasar Framework

Para criar e gerenciar um projeto Quasar, é necessário instalar o Quasar CLI globalmente. Execute o comando a seguir no terminal:

```
npm install -g @quasar/cli
```

Após a execução do comando, o vscode conseguirá lidar com a criação dos projetos em quasar, o que mostraremos a seguir.

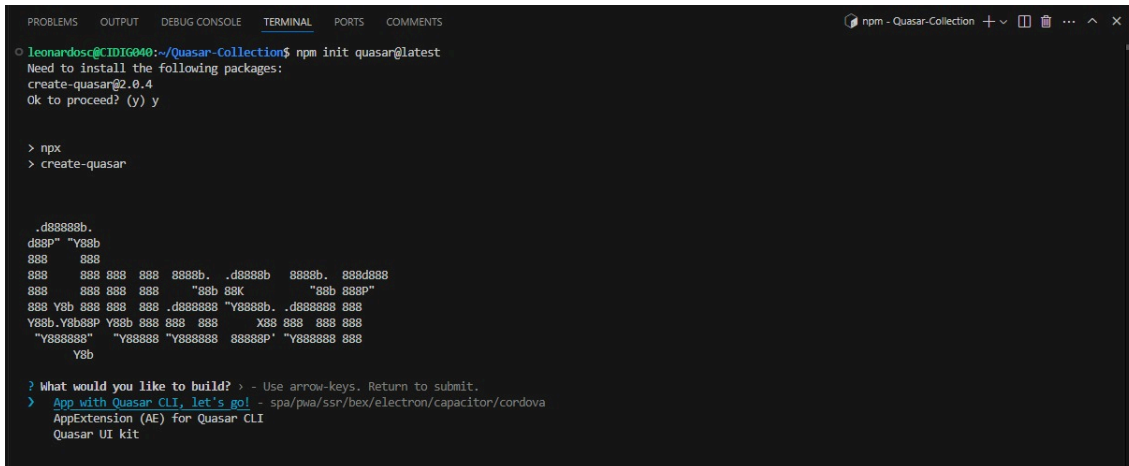
Iniciando um projeto.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS  
● leonardosc@CIDIG040:~/Quasar-Collection$ npm init quasar@latest
```

Com o comando abaixo nós damos início a criação do nosso projeto com a última versão disponível do quasar (atualmente v2.18.1).

```
npm init quasar@latest
```

Escolha da build do projeto.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
o leonardosc@CIDIG040:~/Quasar-Collection$ npm init quasar@latest
Need to install the following packages:
create-quasar@2.0.4
Ok to proceed? (y) y

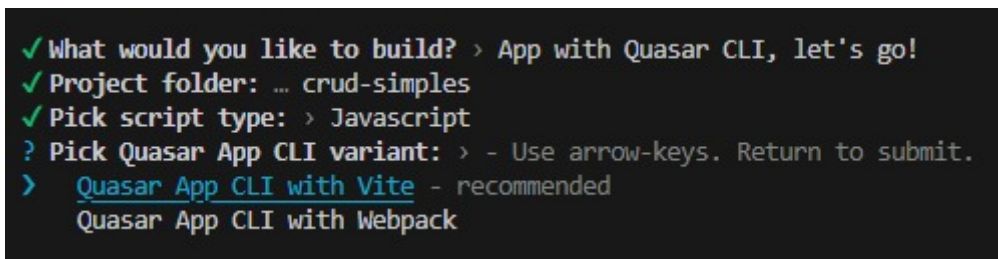
> npx
> create-quasar

.d888888b.
d88P" "Y88b
888 888
888 888 888 888 8888b. .d8888b 8888b. 888d888
888 888 888 888 "88b 88K "88b 888P"
888 Y8b 888 888 .d888888 "Y8888b. .d888888 888
Y88b.Y8888P Y88b 888 888 888 X88 888 888 888
"Y888888" "Y88888 "Y88888P" "Y888888 888
Y8b

? What would you like to build? > - Use arrow-keys. Return to submit.
> App with Quasar CLI, let's go! - spa/pwa/ssr/bex/electron/capacitor/cordova
AppExtension (AE) for Quasar CLI
Quasar UI kit
```

Selecionaremos a primeira opção, pois ela nos possibilita toda a construção do projeto utilizando o Quasar como um framework.

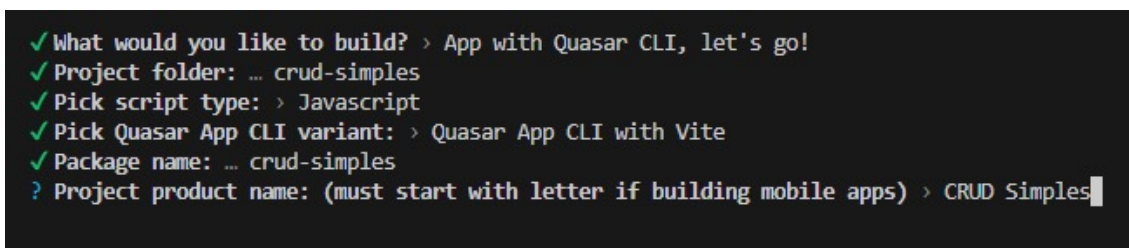
Personalizando as configurações iniciais.



```
✓ What would you like to build? > App with Quasar CLI, let's go!
✓ Project folder: ... crud-simples
✓ Pick script type: > Javascript
? Pick Quasar App CLI variant: > - Use arrow-keys. Return to submit.
> Quasar App CLI with Vite - recommended
Quasar App CLI with Webpack
```

Informaremos um nome para a pasta do projeto (crud-simples), selecionamos qual linguagem trabalharemos no projeto (Javascript ou Typescript) e selecionamos o Vite para o controle do nosso projeto.

Renomeando estrutura inicial.



```
✓ What would you like to build? > App with Quasar CLI, let's go!
✓ Project folder: ... crud-simples
✓ Pick script type: > Javascript
✓ Pick Quasar App CLI variant: > Quasar App CLI with Vite
✓ Package name: ... crud-simples
? Project product name: (must start with letter if building mobile apps) > CRUD Simples
```

Definiremos o nome do pacote (crud-simples) e o nome do produto para caso quisermos transformar em um app mobile (CRUD Simples).

Descrição do projeto

```
✓ What would you like to build? > App with Quasar CLI, let's go!  
✓ Project folder: ... crud-simples  
✓ Pick script type: > Javascript  
✓ Pick Quasar App CLI variant: > Quasar App CLI with Vite  
✓ Package name: ... crud-simples  
✓ Project product name: (must start with letter if building mobile apps) ... CRUD Simple  
? Project description: > Um projeto modelo para roteiro de apresentação do framework
```

Escolhendo o tipo de api do Vue a ser utilizada.

```
✓ What would you like to build? > App with Quasar CLI, let's go!  
✓ Project folder: ... crud-simples  
✓ Pick script type: > Javascript  
✓ Pick Quasar App CLI variant: > Quasar App CLI with Vite  
✓ Package name: ... crud-simples  
✓ Project product name: (must start with letter if building mobile apps) ... CRUD Simple  
✓ Project description: ... Um projeto modelo para roteiro de apresentação do framework  
? Pick a Vue component style: > - Use arrow-keys. Return to submit.  
> Composition API with <script setup> - recommended  
Composition API  
Options API
```

Composition API, pois facilita na construção e interação de componentes e comunicação entre os dados a serem manuseados.

Definindo pré-processadores css.

```
✓ What would you like to build? > App with Quasar CLI, let's go!  
✓ Project folder: ... crud-simples  
✓ Pick script type: > Javascript  
✓ Pick Quasar App CLI variant: > Quasar App CLI with Vite  
✓ Package name: ... crud-simples  
✓ Project product name: (must start with letter if building mobile apps) ... CRUD Simple  
✓ Project description: ... Um projeto modelo para roteiro de apresentação do framework  
✓ Pick a Vue component style: > Composition API with <script setup>  
? Pick your CSS preprocessor: > - Use arrow-keys. Return to submit.  
Sass with SCSS syntax  
Sass with indented syntax  
> None (the others will still be available)
```

Optamos por não configurar nenhum processador devido a simplicidade do projeto, caso o projeto tenha alguma escalabilidade podemos configurar essa parte no futuro sem problemas.

Configurando Features.

```
✓ What would you like to build? > App with Quasar CLI, let's go!
✓ Project folder: ... crud-simples
✓ Pick script type: > Javascript
✓ Pick Quasar App CLI variant: > Quasar App CLI with Vite
✓ Package name: ... crud-simples
✓ Project product name: (must start with letter if building mobile apps) ... CRUD Simples
✓ Project description: ... Um projeto modelo para roteiro de apresentação do framework
✓ Pick a Vue component style: > Composition API with <script setup>
✓ Pick your CSS preprocessor: > None (the others will still be available)
? Check the features needed for your project: >
Instructions:
  ↑/↓: Highlight option
  ←/→/[space]: Toggle selection
  a: Toggle all
  enter/return: Complete answer
  (●) Linting (vite-plugin-checker + ESLint)
  ( ) State Management (Pinia)
  ( ) axios
  (●) vue-i18n
```

Nessa parte, marcaremos o **Linting** e o **vue-i18n** somente para instalação.

- **Linting:** Responsável por verificar e mostrar erros de sintaxe ou variáveis não utilizadas no projeto.
- **State Management:** Controle de variáveis de estado, evita a criação de uma mesma variável em vários componentes.
- **Axios:** Gerenciamento de APIs.
- **Vue-i18n:** Controle de internacionalização.

Configurando formatador de código.

```
✓ What would you like to build? > App with Quasar CLI, let's go!
✓ Project folder: ... crud-simples
✓ Pick script type: > Javascript
✓ Pick Quasar App CLI variant: > Quasar App CLI with Vite
✓ Package name: ... crud-simples
✓ Project product name: (must start with letter if building mobile apps) ... CRUD Simples
✓ Project description: ... Um projeto modelo para roteiro de apresentação do framework
✓ Pick a Vue component style: > Composition API with <script setup>
✓ Pick your CSS preprocessor: > None (the others will still be available)
✓ Check the features needed for your project: > Linting (vite-plugin-checker + ESLint), vue-i18n
? Add Prettier for code formatting? > (Y/n)
```

Adicionaremos o Prettier como modelo para a formatação de código.

Instalando dependências.

```
? Install project dependencies? (recommended) > - Use arrow-keys. Return to submit.  
> Yes, use npm  
    No, I will handle that myself
```

Usamos o npm para instalarmos todas as dependências que selecionamos.

Finalizando instalação e configuração.

```
▼ QUASAR-COLLECTION [WSL: UBUNTU-22.04]  
  > 📁 crud-simples  
  
App • Using quasar.config.js in "esm" format  
App • Generated tsconfig.json and types files in .quasar directory  
App • The app is now prepared for linting, type-checking, IDE integration, etc.  
  
added 497 packages, and audited 498 packages in 32s  
  
128 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities  
  
hint: Using 'master' as the name for the initial branch. This default branch name  
hint: is subject to change. To configure the initial branch name to use in all  
hint: of your new repositories, which will suppress this warning, call:  
hint:  
hint: git config --global init.defaultBranch <name>  
hint:  
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and  
hint: 'development'. The just-created branch can be renamed via this command:  
hint:  
hint: git branch -m <name>  
Quasar • Initialized Git repository 🚀  
  
To get started:  
  
  cd crud-simples  
  quasar dev # or: yarn quasar dev # or: npx quasar dev  
  
Documentation can be found at: https://quasar.dev  
  
Quasar is relying on donations to evolve. We'd be very grateful if you can  
read our manifest on "why donations are important": https://quasar.dev/why-donate  
Donation campaign: https://donate.quasar.dev  
Any amount is very welcome.  
If invoices are required, please first contact Razvan Stoescu.  
  
Please give us a star on Github if you appreciate our work:  
  https://github.com/quasarframework/quasar  
  
Enjoy! - Quasar Team  
  
leonardosc@CIDIG040:~/Quasar-Collection$ cd crud-simples
```

Para navegar até a pasta do projeto usamos o comando abaixo:

```
cd crud-simples
```

Para abrir nosso projeto dentro do usamos o comando **code** seguido do caminho do projeto, neste caso usamos um ponto "." para indicar o diretório atual:

```
code .
```

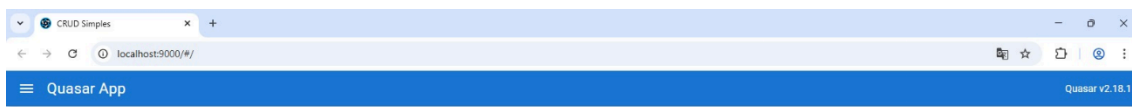
```
● leonardosc@CIDIG040:~/Quasar-Collection$ cd crud-simples  
● leonardosc@CIDIG040:~/Quasar-Collection/crud-simples$ code .
```

Agora, dentro do nosso projeto, podemos usar o comando abaixo para executarmos nosso projeto em um servidor local criado automaticamente:

```
quasar dev
```

```
leonardosc@CIDIG040:~/Quasar-Collection/crud-simple$ quasar dev
```

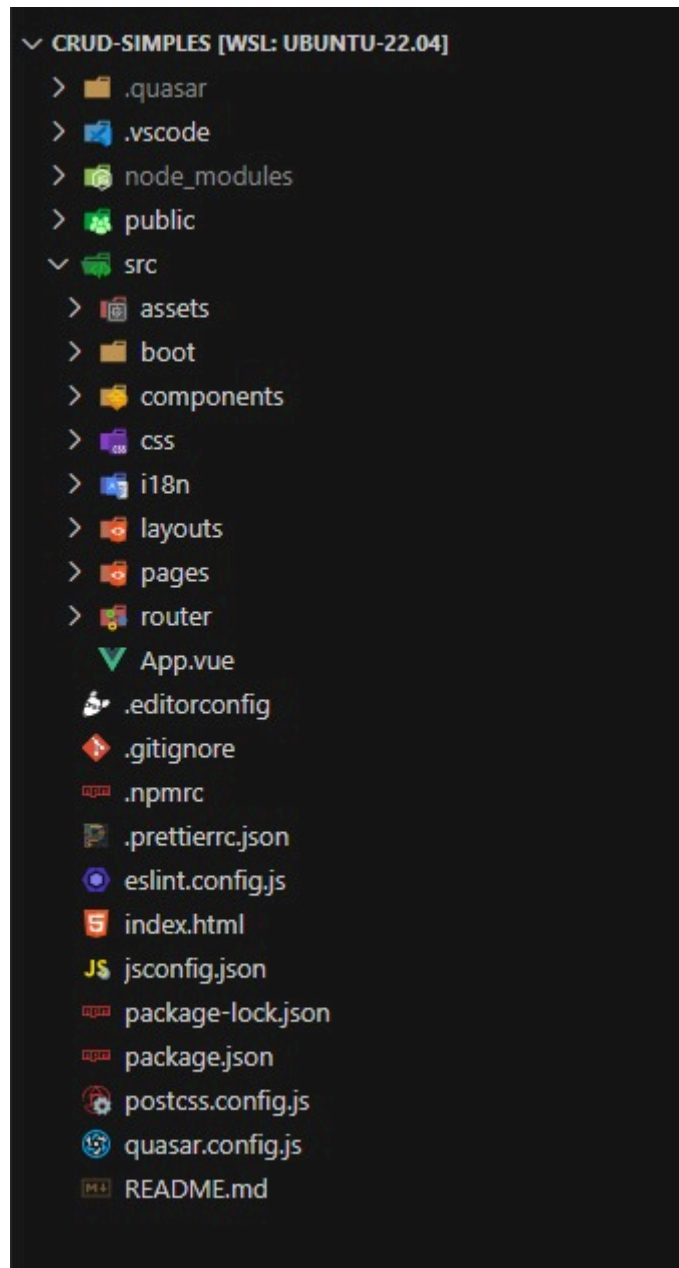
Nosso frontend, por padrão fica acessível no endereço localhost:9000, podem estar em uma porta diferente indicado na saída do comando anterior.



Essa é a interface inicial do nosso projeto.

Entendendo a estrutura de pastas

Iremos focar nas pastas mais importantes do nosso projeto.



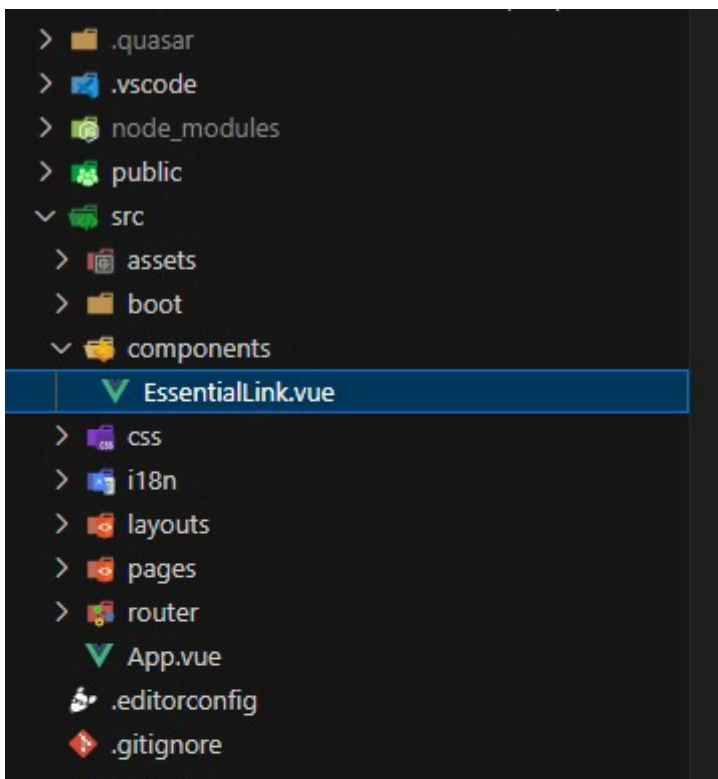
1. **public:** Onde ficam os arquivos estáticos, como favicons ou logos.
2. **src/assets:** Aqui fica as imagens do projeto, ícones, fontes personalizadas.
3. **boot:** Possui os arquivos que precisam ser iniciados antes da inicialização do projeto.
4. **components:** Componentes que são reutilizados em várias áreas do programa.

5. **css**: Arquivos .css ou .scss.
6. **i18n**: Contém arquivos para internacionalização do projeto.
7. **layouts**: Aqui é onde define o layout que a página vai utilizar (cabeçalho, conteúdo e rodapé por exemplo).
8. **pages**: Todas as páginas do projeto.
9. **router**: Onde é feita a configuração e definição de roteamento e navegação do projeto.
10. **App.vue**: Arquivo inicial de um projeto vue, é o arquivo raiz do projeto, que liga todas as comunicações entre os componentes.
11. **index.html**: A página html do projeto, onde toda a lógica do vue é renderizada.
12. **quasar.config.js**: O arquivo mais importante de uma estrutura Quasar, é aqui onde definimos maior parte das configurações que um projeto quasar deve ter. Basicamente, sempre que quisermos adicionar alguma configuração nova iremos alterar esse arquivo.

Criação do projeto CRUD

Para criação do nosso projeto, iremos remover algumas coisas que já vieram configuradas.

Vamo remover o componente **EssentialLink.vue**:



Removemos também a tag que insere a imagem da logo do Quasar no component `IndexPage.vue`:

MainLayout

Aqui, fazemos a alteração do `MainLayout.vue` para somente um header com o título do projeto.

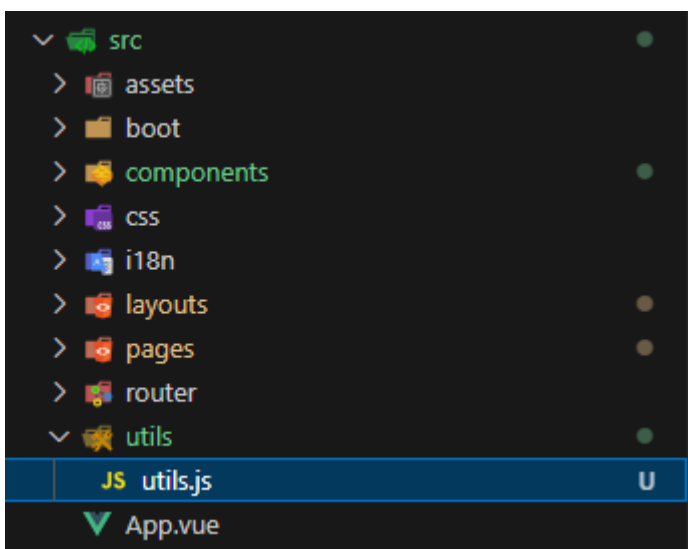
Alteração do RouterMode

Neste moment, alteramos no arquivo `quasar.config.js` a propriedade `vueRouterMode` de “hash” para “history” para mudar a forma do roteamento e remover o “#” do link de navegação:

```
quasar.config.js M X
quasar.config.js > default > defineConfig() callback > build > vueRouterMode
7 export default defineConfig((ctx) => {
21   extras: [
31     'material-icons', // optional, you are not bound to it
32   ],
33
34   // Full list of options: https://v2.quasar.dev/quasar-cli-vite/quasar-config-file#build
35   build: {
36     target: {
37       browser: ['es2022', 'firefox115', 'chrome115', 'safari14'],
38       node: 'node20',
39     },
40
41     vueRouterMode: 'history', // available values: 'hash', 'history'
42     // vueRouterBase,
43     // vueDevtools,
```

Local Storage para guardar os dados

Criaremos em src/Utils um arquivo chamado utils.js onde iremos armazenar as funções que são puramente Javascript para separarmos melhor o código.



Em utils.js irá conter as funções para salvar e ler os dados em localStorage, assim como as funções para editar e excluir.

```
// Salva um value no localStorage com a chave key
function saveLocalStorage(key, value) {
  localStorage.setItem(key, JSON.stringify(value))
}

// Retorna o value salvo no localStorage com a chave key ou um array
vazio caso não exista
function getLocalStorage(key) {
  return JSON.parse(localStorage.getItem(key)) || []
}
```

```

function addNewData(key, data) {
  // Busca os dados salvos no localStorage com a chave key
  const dataStorage = getLocalStorage(key)
  // Gera um id para o novo dado
  const id = dataStorage.length ? dataStorage[dataStorage.length - 1].id
+ 1 : 1
  // Cria um novo dado com o id gerado e os dados passados
  const newData = { id, ...data }
  // Adiciona o novo dado ao array de dados
  dataStorage.push(newData)
  // Salva o array de dados atualizado no localStorage
  saveLocalStorage(key, dataStorage)
}

function editData(key, id, data) {
  // Busca os dados salvos no localStorage com a chave key
  const dataStorage = getLocalStorage(key)
  // Busca o índice do dado que será editado
  const index = dataStorage.findIndex((item) => item.id === id)
  // Substitui o dado antigo pelo novo dado
  dataStorage[index] = { id, ...data }
  // Salva o array de dados atualizado no localStorage
  saveLocalStorage(key, dataStorage)
}

function deleteData(key, id) {
  // Busca os dados salvos no localStorage com a chave key
  const dataStorage = getLocalStorage(key)
  // Busca o índice do dado que será deletado
  const index = dataStorage.findIndex((item) => item.id === id)
  // Remove o dado do array
  dataStorage.splice(index, 1)
  // Salva o array de dados atualizado no localStorage
  saveLocalStorage(key, dataStorage)
}

export { saveLocalStorage, getLocalStorage, addNewData, editData,
deleteData }

```

FormComponent.vue

Iremos criar o componente FormComponent.vue (src/components) que será responsável por ler os dados do usuário e enviá-los ao localStorage.

```

<script setup>
import { addNewData } from 'src/utils/utils'
import { ref, defineEmits } from 'vue'

const emit = defineEmits(['dataUpdated'])

const name = ref('')
const description = ref('')
const telephone = ref('')
const address = ref('')

```

```

const onSubmit = () => {
  const data = {
    name: name.value,
    description: description.value,
    telephone: telephone.value,
    address: address.value,
  }

  addNewData('dataCompany', data)
  emit('dataUpdated')
  onReset()
}

const onReset = () => {
  name.value = ''
  description.value = ''
  telephone.value = ''
  address.value = ''
}
</script>

<template>
  <div class="q-pa-md bg-grey-4">
    <q-form
      @submit="onSubmit"
      @reset="onReset"
      class="q-col-gutter-x-md"
    >
      <div class="row q-col-gutter-md">
        <q-input
          filled
          bg-color="white"
          v-model="name"
          label="Nome *"
          class="col-6"
        />

        <q-input
          filled
          bg-color="white"
          v-model="description"
          label="Descrição *"
          class="col-6"
        />

        <q-input
          filled
          bg-color="white"
          v-model="telephone"
          label="Telefone *"
          class="col-6"
        />

        <q-input
          filled

```

```

        bg-color="white"
        v-model="address"
        label="Endereço *"
        class="col-6"
      />
    </div>

    <div class="q-mt-md">
      <q-btn label="Submit" type="submit" color="primary" />
      <q-btn
        label="Reset"
        type="reset"
        color="primary"
        flat
        class="q-ml-sm"
      />
    </div>
  </q-form>
</div>
</template>

```

TableComponent.vue

Agora iremos criar o TableComponent.vue (src/components) que será responsável por mostrar os dados salvos em localStorage.

```

<script setup>
import { ref, defineProps, watch } from 'vue'
import { getLocalStorage, deleteData } from 'src/utils/utils'

const props = defineProps({
  rows: {
    type: Array,
    required: true,
  },
})

const columns = ref([
  { name: 'id', label: 'ID', align: 'left', field: 'id', sortable: true },
  { name: 'name', label: 'Nome', align: 'left', field: 'name', sortable: true },
  { name: 'description', label: 'Descrição', align: 'left', field: 'description', sortable: true },
  { name: 'telephone', label: 'Telefone', align: 'left', field: 'telephone', sortable: true },
  { name: 'address', label: 'Endereço', align: 'left', field: 'address', sortable: true },
  { name: 'delete', label: 'Excluir', align: 'center', field: 'delete' },
])

const rowsData = ref(props.rows)

```



```

watch(
  () => props.rows,
  (newVal) => {
    rowsData.value = newVal
  },
)

const editar = (id, field, value, scope) => {
  const index = rowsData.value.findIndex((row) => row.id === id)
  rowsData.value[index][field] = value
  localStorage.setItem('dataCompany', JSON.stringify(rowsData.value))
  scope.set()
}

const handleDelete = (key, id) => {
  deleteData(key, id)
  rowsData.value = getLocalStorage('dataCompany')
}
</script>

<template>
  <div class="q-pa-md bg-grey-4 q-mt-md">
    <q-table
      title="Visualização de dados"
      :rows="rowsData"
      :columns="columns"
      row-key="id"
    >
      <template v-slot:body="props">
        <q-tr :props="props">
          <q-td
            v-for="col in columns.slice(0, -1)"
            :key="col.name"
            :props="props"
          >
            <div title="Clique para editar" v-if="col.name !== 'id'">
              {{ props.row[col.field] }}
            </div>
            <q-popup-edit
              v-model="props.row[col.field]"
              v-if="col.name !== 'id'"
              v-slot="scope"
            >
              <q-input
                v-model="scope.value"
                dense
                autofocus
                counter
                @keyup.enter="editar(
                  props.row.id,
                  col.field,
                  scope.value,

```

```

        scope
      )"
    />
  </q-popup-edit>
  <div v-else>
    {{ props.row[col.field] }}
  </div>
</q-td>
<q-td :key="'delete'" :props="props">
  <q-btn
    color="negative"
    label="Excluir"
    @click="handleDelete('dataCompany', props.row.id)"
  />
</q-td>
</q-tr>
</template>
</q-table>
</div>
</template>

```

IndexPage.vue.

Por último, iremos unir esses dois componentes na página inicial em IndexPage.vue.

```

<script setup>
import FormComponent from 'src/components/FormComponent.vue'
import TableComponent from 'src/components/TableComponent.vue'
import { getLocalStorage } from 'src/utils/utils'
import { ref } from 'vue'

const rows = ref(getLocalStorage('dataCompany'))
// Busca os dados do localStorage e armazena em rows

const updateTableData = () => {
  rows.value = getLocalStorage('dataCompany')
  // Atualiza os dados da tabela com os dados do localStorage
}
</script>

<template>
  <q-page padding>
    <FormComponent @data-updated="updateTableData" />

    <TableComponent :rows="rows" />
  </q-page>
</template>

```

Por fim, teremos o nosso projeto funcionando com essa aparência.

CRUD Simples

← → ↺ 🌐 localhost:9000

🔍 ☆ 🏠 ⓘ ⋮

Crud Simples

Nome *

Telefone *

SUBMIT

RESET

Descrição *

Endereço *

Visualização de dados

| ID | Nome | Descrição | Telefone | Endereço | Excluir |
|----|--------|-----------|-----------|-------------|---------|
| 1 | Teste1 | Teste1 | 999999999 | Rua teste 1 | EXCLUIR |

Records per page: 5 1-1 of 1